```
/*****************************************************************

Disclaimer of Warranties
-------------------------
Developed by Dr. Geevar C. Zacharias. You agree that your use of this Software/Service is
solely at your own risk. You agree that such Service(s) is provided on an "as is," "as
available" basis. You are licensed to use, modify, execute and redistribute this software.

Introduction
-------------
In computer science, a disjoint-set data structure, also called a union-find data structure
or merge-find set, is a data structure representing a dynamic collection of sets S =
{S1,...,Sr}. Given an element u, we denote by Su the set containing u. Each set is
identified by a representative, which is some member of the set. It supports three useful
operations:
- MAKE-SET(x): creates a new set whose only member (and thus representative) is x. Since the
sets are disjoint, we require that x not already be in some other set.
- UNION(x, y): unites the dynamic sets that contain x and y, say Sx and Sy, into a new set
that is the union of these two sets. The representative of the resulting set is any member
of this new set.
- FIND-SET(x): returns a pointer to the representative of the set containing x.

Implementation
---------------
This C implementation of Disjoint Set operations are based on trees for a faster
implementation of disjoint sets. Here, we represent sets by rooted trees, with each node
containing one member and each tree representing one set. In a disjoint-set forest, each
member points only to its parent. The root of each tree contains the representative and is
its own parent. Methodology used is union-by-rank heuristics.

Methodology
------------
To implement a disjoint-set forest with the union-by-rank heuristic, we must keep track of
ranks.
- With each node x, we maintain the integer value rank[x], which is an upper bound on the
height of x (the number of edges in the longest path between x and a descendant leaf).
- When a singleton set is created by MAKE-SET, the initial rank of the single node in the
corresponding tree is 0.
- Each FIND-SET operation leaves all ranks unchanged.
- When applying UNION to two trees, we make the root of higher rank the parent of the root
of lower rank.
- In case of a tie, we arbitrarily choose one of the roots as the parent and increment its
rank, we assign the parent of node x by p[x].
- The LINK procedure, a subroutine called by UNION, takes pointers to two roots as inputs.

Program
--------
This program is implemented to accept only integer type of data as dynamic set elements. The
maximum number of singleton set is limited to 7.

Variables (Global)
-------------------
> parent - an integer array to store the parent of each node x.
> rank - an integer array to store the rank of each node x.

Functions
----------
> makeSet(x) - Creates a tree with just one node with only one member x (and thus the
representative).
> findSet(x) - This is a recursive function. We perform a findSet operation by chasing
parent pointers until we find the root of the tree and returns its pointer, which is the
representative.
> unionSet(x, y) - A unionSet operation causes the root of one tree to point to the root of
the other.
```

```
**************************************************************/
#include <stdio.h>

int parent[7];
int rank[7];

void makeSet(int x)
{
    parent[x] = x;
    rank[x] = 0;
}
int findSet(int x)
{
    if(x != parent[x])
        parent[x] = findSet(parent[x]);
    return(parent[x]);
}
void link(int x, int y)
{
    if(rank[x] > rank[y])
        parent[y] = x;
    else
    {
        parent[x] = y;
        if(rank[x] == rank[y])
            rank[y] = rank[y] + 1;
    }
}
void unionSet(int x, int y)
{
    x = findSet(x);
    y = findSet(y);
    link(x, y);
}
int main()
{
    int i;
    makeSet(0);
    makeSet(1);
    makeSet(2);
    makeSet(3);
    makeSet(4);
    makeSet(5);
    makeSet(6);
    //printf("%d", findSet(1));
    unionSet(0, 1);
    //printf("%d", findSet(0));
    unionSet(1, 2);
    //printf("%d", findSet(2));
    unionSet(3, 4);
    //printf("%d", findSet(3));
    unionSet(5, 6);
    //printf("%d", findSet(5));
    unionSet(4, 5);
    //printf("%d", findSet(3));
    unionSet(2, 6);
    //printf("%d", findSet(5));
    for(i=0;i<7;i++)
    {
        printf("Parent of %d is %d.",i,parent[i]);
        printf("Rank of %d is %d.\n",i,rank[i]);
    }
    return 0;
}
```